Part I. Implementation

part 1

```
# Begin your code (Part 1)
"""
face_path: the full path of the folder "face".
list_of_filename: The list of names of the files in the folder "face".

use cv2.imread() to read the images respectively,
and append to dataset, which is the return value.

do the same thing with the folder "non-face".
"""
dataset = []

face_path = os.path.join('.', dataPath, 'face')
list_of_filename = os.listdir(face_path)

for name in list_of_filename:
    re = cv2.imread(os.path.join(face_path, name), cv2.IMREAD_GRAYSCALE)
    dataset.append((re, 1))


non_face_path = os.path.join('.', dataPath, 'non-face')
list_of_filename = os.listdir(non_face_path)

for name in list_of_filename:
    re = cv2.imread(os.path.join(non_face_path, name), cv2.IMREAD_GRAYSCALE)
    dataset.append((re, 0))

# raise NotImplementedError("To be implemented")
# End your code (Part 1)
```

part 2

```python
# Begin your code (Part 2)
"""
in order to preserve bestError and bestClf, I declare them first.
the calculation of Error is refer to the slides teacher gave us.

if Error is smaller the best result I got for now,
let the Error be the new bestError,
and change the corresbing WeakClassifier to bestClf .
"""
bestError = 2e9+1.0
bestClf = WeakClassifier(features[0])
for i in range(len(features)):
    Error = 0.0
    for j in range(len(weights)):
        h = featureVals[i][j]
        if (h < 0) :
            h = 1
        else :
            h = 0

        Error += weights[j] * abs(h - labels[j])

    if (Error < bestError) :
        bestError = Error
        bestClf = WeakClassifier(features[i])

return bestClf, bestError
```

part 4

```python
# Begin your code (Part 4)
"""
tt: a list, stores the lines in the .txt file.
path_list: the full path of the folder "detect".

in the while loop:
  name: the name of the picture.
  num: how many faces are there in the picture.

  I read the same photo twice,
  img is used to draw rectangles on, and
  gray_img is used to do clssification.

  the format of detectData.txt is:
  index of x, index of y, the width(on x-axis), the lenth(on y-axis).

  part_img: the gray, 19*19 image.

  if the classification is "Face", draw a green rectangle, red otherwise.

  I remove the line from tt whenever I've used the information within,
  this could help me ensure the next line I read is a new one.
"""
f = open(dataPath, 'r')
tt = []
for line in f.readlines():
    tt.append(line)

path_list = dataPath.split('/')
sub_path = './' + path_list[0] + '/' + path_list[1] + '/'
```

```python
while len(tt) != 0:
    s = tt[0].split(' ')
    name = s[0]
    num = (int)(s[1])
    img = cv2.imread(sub_path + name)
    gray_img = cv2.imread(sub_path + name, cv2.IMREAD_GRAYSCALE)
    tt.remove(tt[0])

    while(num):
        num -= 1
        s = tt[0].split(' ')
        x = (int)(s[0])
        y = (int)(s[1])
        x_range = (int)(s[2])
        y_range = (int)(s[3])

        part_img = gray_img[y:y+y_range, x:x+x_range]
        part_img = cv2.resize(part_img, dsize=(19, 19), interpolation=INTER_AREA)

        if clf.classify(part_img) == 1:
            cv2.rectangle(
                img, (x, y), (x+x_range, y+y_range), (0, 0, 255), 2)
        else:
            cv2.rectangle(
                img, (x, y), (x+x_range, y+y_range), (0, 255, 0), 2)

        tt.remove(tt[0])

    imshow(name, img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

f.close()
# raise NotImplementedError("To be implemented")
# End your code (Part 4)
```

part 6 (including class, selectbest and classify)

class Classifier (in myclassifier.py)：

```python
class Classifier:
    def __init__(self, feature, standard = 0.0):
        """
          Parameters:
            feature: The HaarFeature class.
            standard: the interger used to classify image
        """
        self.feature = feature
        self.standard = standard
    def __str__(self):
        return "Clf standard = %d, %s" % (self.standard, str(self.feature))

    def modify_standard(self, add_or_sub):
        """
        add_or_sub: True-> add, False-> sub
        """
        if (add_or_sub):
            # self.standard += self.standard/100
            self.standard += 1
        else :
            # self.standard -= self.standard/100
            self.standard -= 1

    def classify(self, x):
        """
        Classifies an integral image based on a feature f
        and the classifiers threshold and polarity.
          Parameters:
            x: A numpy array with shape (m, n) representing the integral image.
          Returns:
            1 if feature(x) < standard
            0 otherwise
        """
        return 1 if self.feature.computeFeature(x) < self.standard else 0
```

selectbest (in myidea.py):

```python
# Begin your code (Part 2)
"""
bestError : The list of number corresponding to bestClf,
            stores the number of wrong classification cases.
now, prev[0], prev[1] : the number of wrong classifications
  now : using current standard
  prev[0] : using the standard that is bigger than current standard
  prev[1] : using the standard that is smaller than current standard

train a classifier for each classifier
adjust standard of current classifier depends on now and prev
return 11 best classifier(I get the number 11 by testing)
"""
bestClf = []
bestError = []

# fixed the size of bestClf and bestError
# which is number of the classifiers we will chose
for i in range(11):
    bestClf.append(Classifier(features[0]))
    bestError.append(2e9)

for i in range(len(features)):
    CC = Classifier(features[i])

    # initialize prev and now
    prev = [0, 0]
    now = 0
    for j in range(len(labels)):
        h = featureVals[i][j]
        now = now + 1 if h < CC.standard else now
        prev[1] = prev[1]+1 if h < CC.standard-1 else prev[1]
        prev[0] = prev[0]+1 if j < CC.standard+1 else prev[0]

    # add_or_sub : True: add, False: sub
    add_or_sub = (prev[1] > now)
    CC.modify_standard(add_or_sub)
```

```
                # set a Maxmum times that standard can be modified
                # in case of oscillating
                COUNT = 0
                while (COUNT < 20):
                    COUNT += 1
                    temp = now
                    CC.modify_standard(add_or_sub)

                    now = 0
                    for j in range(len(labels)):
                        h = featureVals[i][j]
                        h = 1 if h < CC.standard else 0
                        if (h != labels[j]) :
                            now += 1

                    # if now is the best one
                    if now <= prev[0] and now <= prev[1]:
                        break

                    if now > prev[add_or_sub]:
                        add_or_sub = not add_or_sub

                    prev[add_or_sub] = temp

                # find the biggest number in bestError[]
                # and store its index to MAX_BEST
                MAX_BEST = 0
                for k in range(1, len(bestError)):
                    if bestError[k] > bestError[MAX_BEST]:
                        MAX_BEST = k

                # updata the bestClf and bestError
                if now < bestError[MAX_BEST]:
                    bestError[MAX_BEST] = now
                    bestClf[MAX_BEST] = CC

                # for Clf in bestClf:
                #   print(Clf.standard)
            return bestClf
```

classify (in myidea.py)

```python
def classify(self, image):
    """
    Classifies an image by voting,
    if more than half of the classifier classify the image "Face",
    than the image is classify "Face".
      Parameters:
        image: A numpy array with shape (m, n). The shape (m, n) must be
          the same with the shape of training images.
      Returns:
        1 if the image is positively classified and 0 otherwise
    """
    total = 0
    ii = utils.integralImage(image)
    for clf in self.clfs:
        total += clf.classify(ii)
    return 1 if total >= (len(self.clfs)/2) else 0
```

Part II. Results & Analysis:

results:

T = 1

```
Run No. of Iteration: 1
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive regions=[Rectangle
Region(8, 0, 1, 3), RectangleRegion(7, 3, 1, 3)], negative regions=[RectangleRegion(7, 0, 1, 3
), RectangleRegion(8, 3, 1, 3)]) with accuracy: 162.000000 and alpha: 1.450010

Evaluate your classifier with training dataset
False Positive Rate: 28/100 (0.280000)
False Negative Rate: 10/100 (0.100000)
Accuracy: 162/200 (0.810000)

Evaluate your classifier with test dataset
False Positive Rate: 49/100 (0.490000)
False Negative Rate: 55/100 (0.550000)
Accuracy: 96/200 (0.480000)
```

T = 5

```
Run No. of Iteration: 5
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive regions=[Rectangle
Region(10, 8, 1, 1)], negative regions=[RectangleRegion(9, 8, 1, 1)]) with accuracy: 155.00000
0 and alpha: 0.924202

Evaluate your classifier with training dataset
False Positive Rate: 23/100 (0.230000)
False Negative Rate: 0/100 (0.000000)
Accuracy: 177/200 (0.885000)

Evaluate your classifier with test dataset
False Positive Rate: 49/100 (0.490000)
False Negative Rate: 43/100 (0.430000)
Accuracy: 108/200 (0.540000)
```

T = 10

```
Run No. of Iteration: 10
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive regions=[Rectangle
Region(4, 9, 2, 2), RectangleRegion(2, 11, 2, 2)], negative regions=[RectangleRegion(2, 9, 2,
2), RectangleRegion(4, 11, 2, 2)]) with accuracy: 137.000000 and alpha: 0.811201

Evaluate your classifier with training dataset
False Positive Rate: 17/100 (0.170000)
False Negative Rate: 0/100 (0.000000)
Accuracy: 183/200 (0.915000)

Evaluate your classifier with test dataset
False Positive Rate: 45/100 (0.450000)
False Negative Rate: 36/100 (0.360000)
Accuracy: 119/200 (0.595000)
```

T = 1

Analysis:

method 1	train data accuracy	test data accuracy	image 1 accuracy	image 2 accuracy	image 3 accuracy
T = 1	81.00%	48.00%	100.00%	86.67%	50.00%
T = 2	81.00%	48.00%	100.00%	86.67%	50.00%
T = 3	88.00%	53.00%	50.00%	33.00%	50.00%
T = 4	86.00%	47.50%	100.00%	53.33%	50.00%
T = 5	88.50%	54.00%	75.00%	26.67%	50.00%
T = 6	89.00%	51.00%	75.00%	26.67%	50.00%
T = 7	90.00%	54.50%	75.00%	20.00%	25.00%
T = 8	91.00%	55.00%	75.00%	13.00%	25.00%
T = 9	90.00%	57.50%	75.00%	13.00%	25.00%
T = 10	91.50%	59.50%	75.00%	13.00%	0.00%

image 1,2,3 accuracy: the ratio of the faces that are classified correctly.
image1 and 2 are from the folder detect, image 3 is the picture of part 5.

The accuracy of training data increases while T increases. This shows that when T increase, the model will better fit the training data.
But! the accuracy of the image1,2,3 drops when T increases. This shows that the model overfits the data, the model is not that general.
And the test data accuracy increases with T, too. I think that’s because the test data is similar to the training data.

display of part 5:

the result of part 6:

```
Evaluate your classifier with training dataset
False Positive Rate: 17/100 (0.170000)
False Negative Rate: 0/100 (0.000000)
Accuracy: 183/200 (0.915000)

Evaluate your classifier with test dataset
False Positive Rate: 25/100 (0.250000)
False Negative Rate: 23/100 (0.230000)
Accuracy: 152/200 (0.760000)
```

fixed	test condition	train data accuracy	test data accuracy	image 1 accuracy	image 2 accuracy	image 3 accuracy
count < 20, scale = 0.001	member = 5	82.00%	54.50%	25.00%	13.00%	100.00%
	member = 7	84.50%	61.00%	100.00%	26.00%	75.00%
	member = 11	84.00%	62.50%	100.00%	26.00%	75.00%
	member = 13	81.50%	60.00%	25.00%	26.00%	75.00%
scale = 0.001, member = 11	count < 50	84.00%	62.50%	100.00%	26.00%	75.00%
	count < 200	84.00%	62.50%	100.00%	26.00%	75.00%
member = count < 20	scale = 0.01	84.00%	62.50%	100.00%	26.00%	75.00%
	scale = 0.1	84.00%	62.50%	100.00%	26.00%	75.00%
	scale = 1	91.50%	76.00%	75.00%	46.70%	50.00%

above is the progress when I'm testing my parameter of part 6.

count: the maximum times that standard could be modified.

scale: the value standard adds or subtracts each time it is modified.

member: the number of classifiers that are chosen to participate in the voting of classification.

At the end, I choose (count <20, scale = 1, member = 11) as my parameter.

Part III. Answer the questions

1. **Please describe a problem you encountered and how you solved it.**

when I was implementing part 4, I want to cut the image of the face and store it in another variable called "part_image", like it is showed below:

```
part_img = gray_img[y:y+y_range, x:x+x_range]
```

I mistook the order of the parameters, I thought it would be x to be the first parameter, but it turns out to be y.

Since the "part_image" is used in clf.clasify(), which is separated from the one I use to draw red/green rectangle on, there was no error in the first picture (the-beatles.jpg). The error came out when dealing with the second picture (p110912sh-0083.jpg).

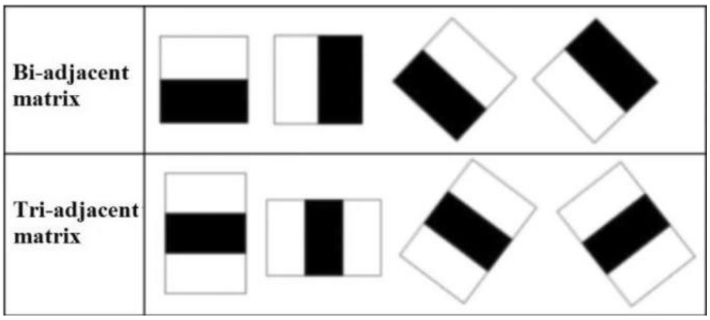
Through my observation and some test, I found that whenever the index of x is above 1200 there will be an error. I went to check the size of the picture, and it seemed that everything was fine. Then I went to HW1_Discussion to see if anyone else has the same problem with me, and answer is no. So, I checked the syntax of cutting image again, and found the correct syntax.

2. **What are the limitations of the Viola-Jones' algorithm?**

- Restricted to binary classification
- May be sensitive to very high or low exposure.
- Mostly effective when face is in frontal view
- It will take lots of time summing up pixel values for all feature types in all images in the dataset.

3. **Based on Viola-Jones' algorithm, how to improve the accuracy except increasing the training dataset and changing the parameter T?**

We can expand Haar-like rectangle feature by adding 45°-rotatable rectangle feature. Like it is shown below:



https://jivp-eurasipjournals.springeropen.com/articles/10.1186/s13640-019-0435-6

Through the expand Haar-like feature, more details of faces can be represented, and thus the accuracy could improve.

Or we can train some classifiers to detect eyes, nose, mouth… and so on. Then by determine whether there are eyes, nose (or other facial features) separately, to decide whether this is a human face or not. It doesn't have to meet all the facial features to be classify as "face", maybe two or three is enough(I think that will be the parameter we have to find out).

By classifying facial features respectively to classify human face can be more precise than the original one, so I think it could improve the accuracy. (though it might be time consuming)

4. Please propose another possible face detection method (no matter how good or bad, please come up with an idea). Please discuss the pros and cons of the idea you proposed, compared to the Adaboost algorithm.

We can use template matching. By using pre-defined or parameterized face templates to locate or detect the faces by the correlation between the templates and input images. A face model can be built by edges just by using edge detection methods.

pros: The implement of this approach is simpler than Adaboost algorithm.

cons: Since everyone has different face contour, different types of nose, eyes and mouth, this method may fail to detect the person with facial features that are not in my templates.